LAB 1: GRAPHICAL MODELS

Namita Sharma

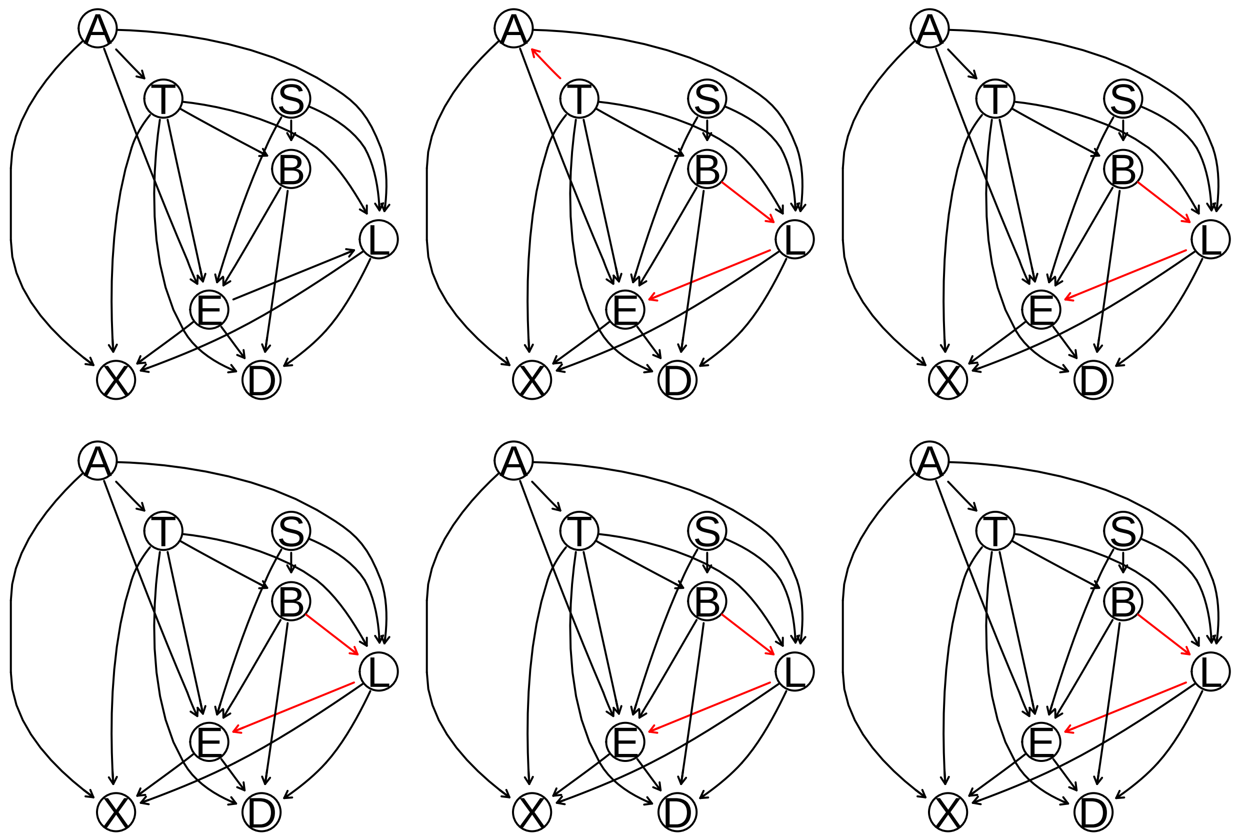
9/11/2020

1. Hill-Climbing to learn the BN structure

```r
library("bnlearn")
library("gRain")

# Q1----------------------------------------------------------
data("asia")
n = nrow(asia) # number of samples

# Multiple runs of Hill Climbing to learn the BN structure
bn1  <- list()
comp <- c()
for (i in 1:6) {
  bn1[[i]] <- hc(x=asia, score="bde", iss=100, restart=10)

  if(i!=1)
    comp[i] <- all.equal(bn1[[i-1]], bn1[[i]])
}

# Comparison of different BN structures learnt
print(comp)
```

```
## [1] NA
## [2] "Different number of directed/undirected arcs"
## [3] "Different arc sets"
## [4] "Different arc sets"
## [5] "TRUE"
## [6] "TRUE"
```

```r
par(mfcol = c(2,3))
do.call(graphviz.compare, bn1)
```



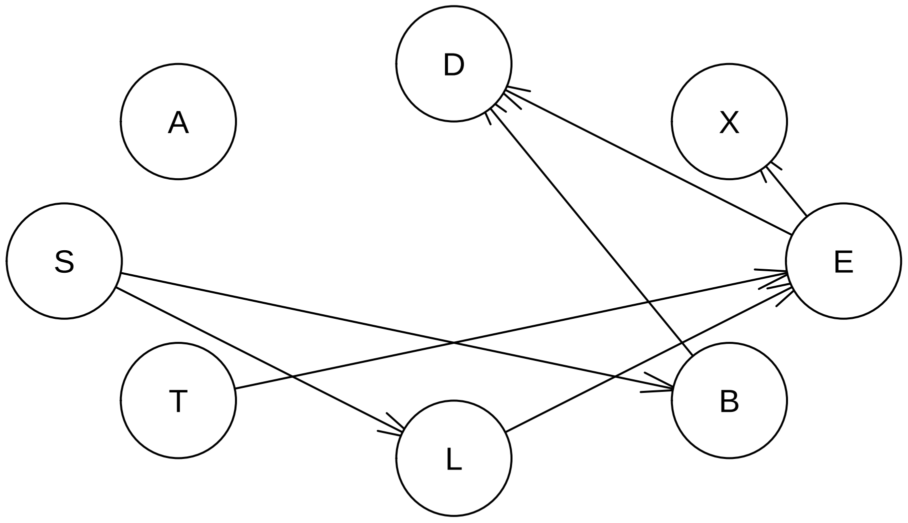
The HC algorithm was run 10 times with hyperparameters score=BDeu, imaginary sample size=100 and number of random restarts=10. The different outputs were compared for equivalence. It was seen from the comparisons that the algorithm can learn non-equivalent Bayesian network structures for the same data in different runs. This can be explained from the stochastic and heuristic nature of the HC algorithm. In each iteration it starts at a random initial state and explores possibly different sets of neighbouring states and follows different optimization paths through the random restarts. Hence it may result in different BN structures.

2. Fit network parameters and predict from the structure of the BN

```r
# Q2-------------------------------------------------------
# 80-20 split of data for train-test
set.seed(12345)
index <- sample(1:n, n*0.8)
asia_train <- asia[index, ]
asia_test  <- asia[-index, ]

# Function to fit the parameters of BN and predict S based on the structure of BN
predict_S <- function(bn, evidence_nodes, predict_node, data_train=asia_train,
                      data_test=asia_test) {

  # Fit the parameters of the BN conditional on its structure
  bn_fit <- bn.fit(bn, data=data_train, method="mle")

  # Convert fit object to grain object
  bn_grain <- as.grain(bn_fit)
  # Prediction of S
  n_test <- nrow(data_test)  # Number of test samples
  S_predict <- character(n_test) # Vector to hold predictions of S

  for (i in 1:n_test) {
    # Set the evidence i.e. the state of all other nodes except S
    evidence = setEvidence(bn_grain,
                           nodes=evidence_nodes,
                           states=sapply(data_test[i, evidence_nodes], toString))

    # Conditional distribution of S given all the other nodes
    S_cond = querygrain(evidence, nodes=predict_node, type="conditional")

    # Classify S as yes or no depending on the conditional posterior of S
    if (S_cond[1] > S_cond[2]) {
      S_predict[i] = names(S_cond)[1]
    } else {
      S_predict[i] = names(S_cond)[2]
    }
  }

  # Compute the confusion matrix and misclassifcation rate
  conf_mat = table(S_predict, data_test[, predict_node])
  mc_rate = 1 - (sum(diag(conf_mat)) / sum(conf_mat))

  return(list(confusion_matrix=conf_mat, mc_rate=mc_rate))
}

# Learn the BN structure from train dataset
bn2 <- hc(x=asia_train)
print(bn2)
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##    [A][S][T][L|S][B|S][E|T:L][X|E][D|B:E]
##   nodes:                                 8
##   arcs:                                  7
##     undirected arcs:                     0
##     directed arcs:                       7
##   average markov blanket size:           2.25
##   average neighbourhood size:            1.75
##   average branching factor:              0.88
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 BIC (disc.)
##   penalization coefficient:              4.147025
##   tests used in the learning procedure:  77
##   optimized:                             TRUE
```

```r
plot(bn2)
```



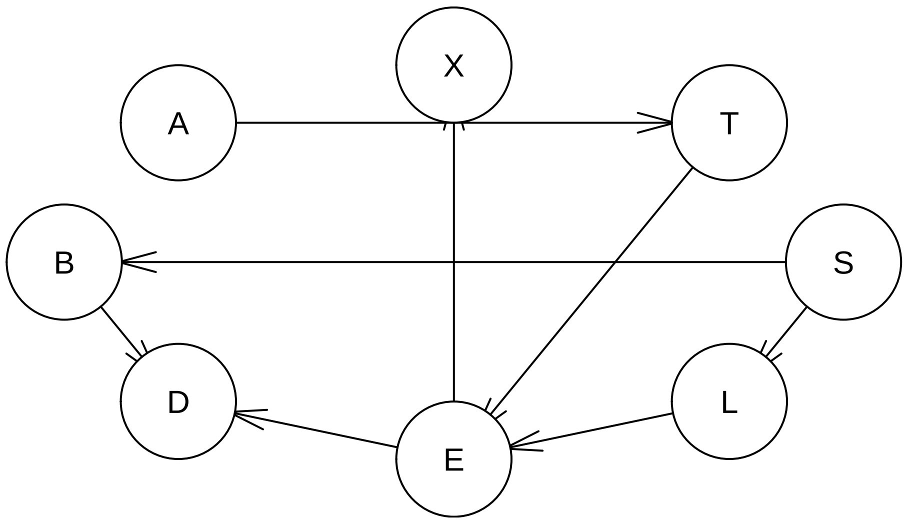
```
# Predict S based on the learned BN structure
S_predict = predict_S(bn=bn2,                             # Learned BN
                      evidence_nodes=names(asia)[-2], # All nodes except S
                      predict_node=names(asia)[2])    # Node S
print(S_predict)
```

```
## $confusion_matrix
##
## S_predict  no yes
##       no  337 121
##       yes 176 366
##
## $mc_rate
## [1] 0.297
```

```
# True Asia bayesian network
dag <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
print(dag)
```

```
##
##   Random/Generated Bayesian network
##
##   model:
##    [A][S][B|S][L|S][T|A][E|L:T][D|B:E][X|E]
##   nodes:                                 8
##   arcs:                                  8
##     undirected arcs:                     0
##     directed arcs:                        8
##   average markov blanket size:            2.50
##   average neighbourhood size:             2.00
##   average branching factor:               1.00
##
##   generation algorithm:                   Empty
```

```
plot(dag)
```



```
# Predict S based on the true BN structure
S_predict_true = predict_S(bn=dag,                               # True BN
                           evidence_nodes=names(asia)[-2], # All nodes except S
                           predict_node=names(asia)[2])     # Node S
print(S_predict_true)
```

```
## $confusion_matrix
##
## S_predict  no yes
##       no  337 121
##       yes 176 366
##
## $mc_rate
## [1] 0.297
```

The confusion matrix of predictions using the learnt bayesian network and that of the true Asia network was found to be the same. This can be explained by looking at the two graph structures. While the learnt BN may not be equivalent to the true Asia network i.e. has different number of directed arcs/edges and orientations, but in terms of the predicted node S, the independencies in the true network have been learnt quite accurately from the data. The learnt network structure also encodes the same independencies as that of the true network in terms of node S. Hence the posterior distribution of S cam be said to be close to the actual distribution of S.

3. Classify based on Markov-blanket

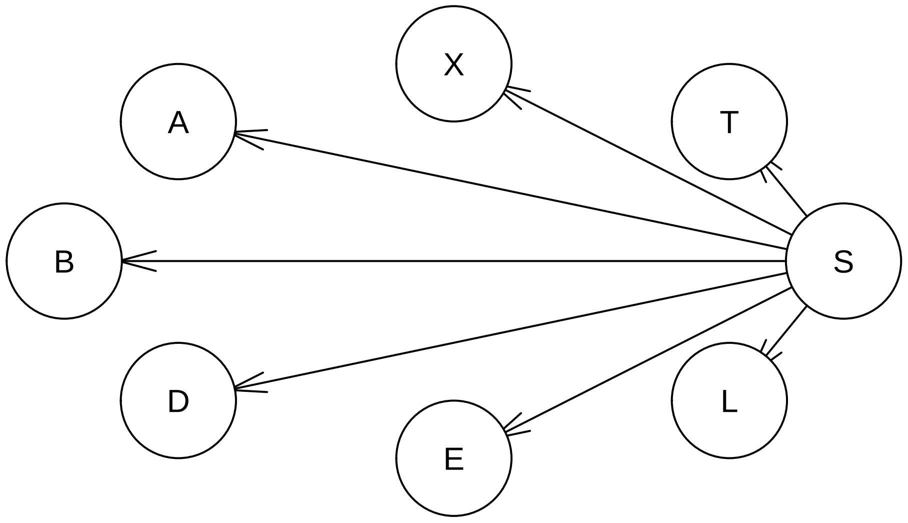
```r
# Q3--------------------------------------------------------
S_predict_mb = predict_S(bn=bn2,                                # Learned BN
                         evidence_nodes=mb(bn2, node="S"), # markov blanket of S
                         predict_node=names(asia)[2])      # Node S
print(S_predict_mb)
```

```
## $confusion_matrix
##
## S_predict  no yes
##       no  337 121
##       yes 176 366
##
## $mc_rate
## [1] 0.297
```

Again, the confusion matrix of predictions using the markov blanket of S was found to be the same as that of the learnt bayesian network. This can be explained because given a node, its markov blanket encodes the independencies w.r.t to that node in the network i.e. given the nodes L and B (the markov blanket of S) in the network, S becomes independent of the remaining nodes in the network. Hence the predictions and the accuracy obtained by conditioning on the markov blanket of S is the same as the predictions obtained by conditioning on all the nodes of the learnt bayesian network.

4. Naive-bayes classifier

```r
# Q4--------------------------------------------------------
# Naive-bayes classifier
nb = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
plot(nb)
```



```r
S_predict_nb = predict_S(bn=nb,                                    # Bayes-classifier
                         evidence_nodes=names(asia)[-2],   # All nodes except S
                         predict_node=names(asia)[2])      # Node S
print(S_predict_nb)
```

```
## $confusion_matrix
##
## S_predict  no yes
##       no  359 180
##       yes 154 307
##
## $mc_rate
## [1] 0.334
```

The confusion matrix of predictions using the naive-bayes classifier was found to be different from that of the learnt bayesian network. Overall, the mis-classification rate was found to increase/worsen with the naive-bayes classifier. This could be due to the naive assumption that the NB classifier makes- that all the other nodes are independent of each other given S. Whereas we know from the true network that some of these independencies are not true. So, the NB classifier encodes some additional (false) independencies in the graph structure that result in poorer predictions for S.